Nuna Evolution

José van Amson, Kees Boon, Chi-Yuen Pang

August 31, 2010
# Contents

1 Assignment 6  
   1.1 Original Game 6  
   1.2 Goal and conditions 6  

2 Orientation and Plans 8  
   2.1 Orientation 8  
   2.2 Gameplay plans 10  
      2.2.1 Gameplay plans 10  
      2.2.2 Game Flow 11  
      2.2.3 Challenges 11  

3 Development 13  
   3.1 Team 13  
   3.2 Agile Development 13  
   3.3 User Tests 14  
      3.3.1 First Test Session 14  
      3.3.2 Second Test Session 17  

4 Architecture and Implementation 19  
   4.1 Architecture Overview 19  
      4.1.1 Original Architecture 19  
   4.2 Component description 21  
      4.2.1 Client-Server model and Network 21  
      4.2.2 CS-model task delegation 22  
      4.2.3 Racegame (server) 23  
      4.2.4 Designer (Client) 25  
      4.2.5 Both 26  

5 Assets 28  
   5.1 2D Assets 28  
   5.2 3D Assets 29  
   5.3 Audio Assets 29
## Final Product

6.1 Race game ................................................. 31
6.2 Designer .................................................. 31
6.3 Unachieved goals ........................................ 32

## Deployment

7.1 Set-up ..................................................... 33
7.2 Technical difficulties .................................... 33
    7.2.1 Controller connectivity problems ................. 33
    7.2.2 Full screen problems ............................... 34
    7.2.3 Solutions to deployment problems ............... 34

## Recommendations

8.1 Physics .................................................. 35
8.2 3D-Models ............................................... 35
8.3 Multiplayer ................................................. 35
8.4 Multiple track and car choice .......................... 36

## Bibliography

37

## Appendix

A 38
A.1 Assignment ............................................... 38
A.2 Plan of Approach ........................................ 38
A.3 Game Design Document ................................. 38
A.4 Simplified object diagram of the game ............... 38
The bachelor project consists of a internship and our project was executed in commission of the Science Centre TU Delft and Tinker Imagineers bv. Our assignment was to further develop on a previous project that was selected to be showcased at the Science Centres Gamelab.

The game in question is the Nuna Racing game, where a team of two players (a driver and a designer) try to achieve the highest score while driving a lap at the Zandvoort racetrack. While driving the designer can change aspects of the car such as the body, wheels and engine. Each choice has its own advantages and disadvantages but the choice also has an impact on the score.

The game was already fully functional, but it offered a lot of room for expansion. The biggest priority was not to lose the Serious Gaming aspects while making the game more fun to play. We achieved this by improving the feedback and interaction with the players. It was accomplished by introducing sound to the game and user interfaces. Interaction between players was achieved by splitting and switching information between the driver and designer screens, both players communicated a lot more in order to exchange information.

To make the game more attractive and challenging, we introduced challenges throughout the track. When challenges are successfully completed a number of credits will be earned, these credits can be used to enhance parts of the car making a higher score possible. The highest scores are displayed at the end of the lap, introducing a competition aspect to the game.

Finally the game has been deployed on one pc, having two separate programs for each player. The two programs form a client-server structure, with the racer as the server and the designer as the client. The driver has a large screen and the designer is equipped with a normal sized touch-screen. The driver has a Xbox steering wheel and pedals for driving the car.
Introduction

The bachelor project consists of an internship at which a group of students can put the knowledge and experience learned in past courses into practice. The goal is to gain experience in realistic projects, including the whole software design process from design to delivery. The project consists of analyzing the given task, studying possible solutions, selecting the solution most suited to the problem and implementing it. The implementation must be tested and delivered conforming to task specifications.

This report is the final report for the bachelor project for the Computer Science course. The project has been executed in commission of Roel Bolhuis from Tinker Imagineers bv and the Delft Science Centre. We are very grateful for the aid and guidance through the project. We would also like to thank our counselors at the Delft University, Drs. P.R. van Nieuwenhuizen, Dr. ir. A.R. Bidarra, Bert Belder and Arne Bezuijen for the guidance and assistance during the whole project.

Context and motivation

The Science Centre TU Delft is a part of the Delft University and it is opening its doors in September 2010. It will function as a platform for students, scientists and the public. One of its categories is the Game Lab, where visitors can see Serious Games being developed and actually play them too. One of these games is the Nuna Racing Game. It started as a project for the Building Serious Games course and has been chosen to be displayed in the Game Lab, however the game was not completely ready for deployment. Finishing the game became a task offered as a bachelor project. This task offered the freedom to complete the game in any way as long as the Serious Game aspect remained intact or was improved. The previous team delivered many suggestions on how to improve the game and two members of the previous team were also prepared to help with the further development of the game.
Report overview

First we will describe the task and its specifications in chapter 1. In chapter 2 we will describe the game design and the planning followed by a description of the development process in chapter 3. After that we will discuss the implementation and design choices in chapter 4. Following that, chapter 5 focuses on the game assets. The resulting game is discussed in chapter 6. Finally the deployment will be discussed in chapter 7. Some recommendations are given in chapter 8.
Chapter 1

Assignment

The assignment consists of further developing the prototype version of the Nuna Race Game into a full playable release version in the course of 12 weeks. The developers have been given the freedom to make many design options and add any desired aspect to game in order to improve the Serious Gaming concept and making the game more appealing to the crowd.

1.1 Original Game

The game started out as a project for the Building Serious Games course. The team consisted of Bert Belder, Arne Bezuijen, Lennard de Rijk, Bas van Nuland, Nick Kraayenbrink and Roelof van den Berg. A serious game is a game whose primary purpose is anything other than entertainment but more focused on education. The purpose of the original game is to teach the players about the impact of different car parts on the performance of the car and its impact on the environment.

The original game we started out with consisted of two segments, the driver and the designer. The driver drives the Nuna5 car around the Zandvoort track and the designer changes the car during the race. The goal was to make a lap as fast and environmental-friendly as possible.

1.2 Goal and conditions

The conditions of the project were short and simple. The serious game aspect of the game needs to be clearly present in the game, the game itself needs to be robust in order to withstand the continuous running times and the game needs to be more appealing for the crowd. Given these conditions, we set up goals in order to lead the project to a successful game release. These goals are:

- Increase interaction between player/game and player/player
• Increase the feedback provided by the game

• Improve user friendliness

In the following chapters we will discuss what we have done in order to reach these goals by first describing our orientation and the gameplay plans we obtained from that, followed by the development methods, architecture and the assets of the game. Finally we will describe the final product and how is has been deployed.
Chapter 2

Orientation and Plans

First we have oriented ourselves regarding stakeholders of the assignment; what they do and what they expected from us to make and deliver for this project. It is a project where the developers can get lost in their own creativity. That is why we had to find and define a way to reach the end of the project.

Much of this information is written in our plan of approach and as such a significant portion of it will be largely repeated or paraphrased in this chapter.

2.1 Orientation

Stakeholders

The concerned users are, the Science Centre, and by extension the TU Delft. The current year project members: José van Amson, Chi-Yuen Pang and Kees Boon. The members from the previous year: Bert Belder and Arne Bezuijen. Rafael Bidarra provides oversight. Finally, Tinker Imagineers through Roel Bolhuis. The current year members will do the programming for extending the original game. The previous year members should offer assistance with technical issues that may arise and provide graphical support. End users for the game are Delft Science Centre visitors, so the game should be playable by a wide variety of users. Most prominently the game is focused on young adolescents.
Main assignment originator: Science Centre Delft

Project manager: Roel Bolhuis from Tinker Imagineers
mobile +31(0)616118719
roel@tinker.nl
Nieuwegracht 13. 3512 LC
Utrecht

TU Delft oversight: Rafael Bidarra
phone +31152784564
R.Bidarra@tudelft.nl
Mekelweg 4. 2628 CD
Delft

Graphics Support Bert Belder
mobile +31(0)625381064
h.belder@student.tudelft.nl

Arne Bezuijen
mobile +31(0)618812753
arne@arnage.nl

Project members Chi Yuen Pang
mobile +31(0)648262947
chiyuenpang@hotmail.com

José H. van Amson
mobile +31(0)622500059
jhvanamson@gmail.com

Kees Boon
mobile +31(0)618444705
boon.kees@gmail.com

Problem definition
The Science Centre requires further development of a Serious Game for the purpose of showcasing it on the Serious Game exhibition in the GameLab at the Delft Science Centre. At the outset of the project there was only a basic racing game with a few design options, this has to be extended.

Goal
Extend the existing race game so that it becomes a better learning experience that is still fun. It should be playable in a two player race game setup, in
which one player (driver) uses an Xbox wheel and pedals to race while the
other player (designer) uses a touch-screen to change the settings of the
car. Different settings have different effects on the cars’ behavior, where
each setting has certain advantages and disadvantages. The game should
emphasize the design-while-you-race aspect yet still be enjoyable for both
partaking players.

Conditions

The following conditions applied to the creation of the game:

- Serious Game concept must be pronounced
- Hardware (2 players, steering wheel, touchscreen)
- Robustness
- Score system must be improved
- Improve game as much as possible

2.2 Gameplay plans

The gameplay plans we made during the orientation phase can be summa-
rized by the bullets listed below. Again we paraphrase from the plan of
approach document. These were just based on the crudely conceived ideas
of gameplay issues, with no implementation yet in mind.

- Scoring system
- Design choices
- Challenge/Reward system

2.2.1 Gameplay plans

The game is a two player game. Each player has a role; one player is the
driver, the other is the designer. The goal is to complete a single lap around
the game’s track, and thus obtaining a high score. Besides lap time, the score
is dependent on several other things. Both player roles and their interaction
are described below. As written in our Game design document paragraphs
game flow, Design, challenge, scoring factor and game sound will be repeated
in this chapter.
Driver

The driver’s responsibility is completing a lap in as quick a time as possible, while also remaining inside the track. In doing so, the driver can complete racing challenges that provide credits, which may be used by the designer. The challenges are detailed below. How well the driver is able to race around the track depends on the driver’s own skill, but also on the designer.

Designer

The designer’s responsibility is to modify the racing car while the race is underway. There are no constraints on the choices the designer has in this. However, the credits mentioned in driver may be used to do upgrades. Upgrades can be applied to components that the designer can choose. The components that can be chosen and their effects are described in paragraph design. Additionally, the designer can make a design prior to the start of the race, where a few upgrade credits are available. One-off design parameters are available at this phase.

Score

Obtaining a high score does not solely depend on the fastest lap, several engineering parameters, such as environmental impact and ease of production are taken into account when computing the score.

2.2.2 Game Flow

Below the states of the game as it is played in nominal order are listed.
For a more detailed description the user is referred to the plan of approach, though the meaning behind the states should be intuitive.

1. Start state
2. Tutorial state
3. Initial design state
4. Racing state
5. Highscore state

2.2.3 Challenges

The racetrack contains five types of challenges: bonus strips, a time trial, a slalom, staying within the track for a certain time, and taking a bend in the ideal line. The first three are explicit and the latter are awarded when they are completed but not explicitly communicated to the players in advance. A
short description of each follows. Again note that points denote what must be obtained for a highscore, while credits are used to purchase upgrades. Challenges are set up in such a way that they are indeed challenging but are made easier with proper design choices. This should enhance the emphasis on engineering choices and thinking ahead and should provide an incentive for player cooperation.
Chapter 3

Development

3.1 Team

The project development team consists of two Computer Science students of the Media and Knowledge Technology variant and one Computer Science student of Systems Engineering variant. José van Amson, Kees Boon and Chi-Yuen Pang respectively. Two students from the former project provided us aid and support during the project, namely Bert Belder and Arne Bezuijen, both Industrial Design students. Roel Bolhuis was the supervisor on behalf of Tinker Imagineers bv and provided us of logistical support. Dr. ir. A. R. Bidarra, on behalf of the Technical University of Delft, provided us of technical supervision through the project.

3.2 Agile Development

We decided to take an informal approach on the work process since the project team was relatively small, working full time and in the same room. By dividing tasks in clusters and planning them we created an overview of what needed to be done and when it had to be finished. This approach gave us the freedom to assign clusters to group members yet at the same time keep the flexibility in taking over each others clusters. Meetings were substituted by multiple short discussions. These discussions took place whenever a milestone was hit, problems had to be solved or when the overview was threatened. This resulted in minor adaptations to the planning, making it possible to complete the project in 12 weeks.

In conclusion, our work process has been quite informal, but at the same time very powerful and agile. We were capable of dealing with problems and adapting our plans to meet the deadline.
3.3 User Tests

Testing the game is a necessary step to get an objective feedback from the end-users. It us correct our biased view of the game and it also provided ideas for new improvements. During development we underwent two test sessions, even though we would have liked to do more, circumstances (some of which were our own fault) dictated otherwise.

3.3.1 First Test Session

In the first test session with the children, we used two laptops where one of the laptops ran the game without challenges and the other laptop used a race game with latest challenges. The purpose of the first test is to get our first experience with end-users and how they react to the game and to the handling of the car.

Overview

The first test suffered from some poor conditions. Due to a error we introduced (unintentionally of course) into the code shortly before the testing took place, the car in the race game would come to a sudden and complete stop every time a design choice was made by the designer. This bug strongly affected the gameplay in a bad way.

The test was ran with roughly two dozen young boys aged about 6 though 8 years old. This was unexpected, as we had anticipated a slightly older target group. We had two laptops that were able to run the game, and two Xbox controllers to stand in for the steering wheel that the final setup would have. One of these controllers was kindly provided by the Robot Pacman development team, with whom we shared the room. In addition to the laptop screens, we had secondary monitors connected to the laptops, so the race game and the designer could be run in separate screens.

Each testing session was started with a short explanation of the game, followed by test sessions, in which we made notes and sometimes assisted when technical problems occurred. Two players would simultaneously play the game with one player racing and the other designing.

Overall the test went quite chaotically, because we were ill prepared to manage so many children and the speed and conditions of the tests. The main things we learned about the game were firstly that despite the crippling bug, players still found the game enjoyable. Because of the enthusiasm we observed, we believed the positive feedback to be genuine and not just because of politeness. Secondly, we were able to identify several icons in the designer screen that were unclear, and some that were intuitive. As a consequence, we reworked the unclear icons and decided on text support for each icon as well. A third point was that the real-time score gaining system,
analogous to what one might see in an arcade game, was almost entirely ignored. This lead us to remove that feature altogether. We also decided to liven up the game world with additional artwork.

The feedback we obtained are listed below in the order of remarks made by the players during the game, our own observations and some interview questions we asked after the game.

Remarks

The players provided the following remarks:

- Design is nice.
- Racing is too difficult.
- It will be nice to have sound.
- Graphics are too simple.
- It is funny to play but you can not do a lot.
- Coloring the car is fun.
- It is annoying when changing the car settings will cause the car to stop.
- Playing designer is fun.
- Racing is more fun.
- When you find good car setting then the car driving well.
- Having a pit stop in game is cool.
- Circuit is too long.
- It is harder to steer on grass then on track.
- Challenges are too difficult. (Test with challenges)
- It is more fun than a normal race game. (Test with challenges)
- It is emptier then a normal race game.
- It will be more fun if I can play with a steering wheel.
Observations

- The purposes of icons or buttons are not always clear to the children.
- There are big differences between ability of the children to use the gamepad to drive the car.
- For some children the racing with the gamepad is too difficult, but some have hardly any difficulties with steering with a gamepad.
- Hard to find a good car setting.
- Score points on screen is often confused with the speed of the car.
- The gamers don’t know what the material icons on the designer screen would do when they click on them.
- The designer usually looks on the race screen en not on the designer screen because there is not much interaction in the designer screen.
- Signs for approaching a corner are confused with an assignment of turn the car backward.
- There is a bug in the setting update which causes the car to stop when the car setting is changed.
- The car is sliding too easily.
- The meaning of the stars which are for part upgrade is not clear.
- There is a bug with changing the car color.
- Curbs are too high. It got too much effect on the car, it sometime cause the car to slip.
- Race challenges make het game run slow.
- Children are delighted they can play with a car they just have seen in the science center.
- The co-operation between the racer and designer has a big effect how much fun the players have. If the racer is dominant then the designer get task by the racer. If the racer and designer are working like a team they have more discussions about improving the car.
- The children who are playing computer/console games are comparing with the race game they know.
- The mini-map is confusing for some of the children.
- The green icons like solar panels made children aware that the game was influenced by environmental effect.
Points to improve upon

From the interviews and our own discussion of the test the following points came to our attention.

- Button and icon should have text to explain the user what it means and what it does.
- Changing the car physics to make it more playable for children with no computer game experience.
- Fix the bug which affects the car speed when the designer changes a car part.
- Remove the score and show the score at the end.
- Change the score function.
- Lowering the curb and the track level.
- Changing the icon of materials on the designer screen.
- Moving the signs like approaching a corner from race screen to designer screen.
- Fix the performance problem with challenges.
- Try to increase the co-operation between the racer and designer.
- Improve mini-map layout so the gamers know they looking on mini-map of the racetrack.

3.3.2 Second Test Session

In the second test session we also used two laptops to test the game. But this time both games will be tested with game challenges. The purpose of the second test is find out how the users are reacting to the challenges and if we need some more tweaks on the car physics.

Overview

Again we had two laptops with controllers and additional screens available. Each session was run in the same way as the previous user test.

Because of our experience with testing with a product that had a critical bug, we had a more stable version on hand before the second test commenced. Due to poor communication with the Science Centre staff however, we had assumed the test to take place in the late afternoon. Instead it was scheduled earlier, because the players were already present in the building, and had to leave shortly after the test. As a result the test was hurried and
we obtained little results. Mainly we were reinforced in our ideas to provide ingame feedback based on the players actions and improve the car control.

**Observations**

- The gamers are not sure what selecting another car part will do.
- Turning the car is difficult. For example when the car skidded and turned backward, it became very hard to turn the car and face the right direction of the track.
- Effect of changing car setting is not always noticeable.
- The purpose of the challenges are not clear to the players.
- The car can still can get stuck on the track.

**Points to improve upon**

- Adding sounds when the car settings are changing
- Reduce the turning circle of the car.
- Implementing a feature which moves the car back on track.
- Provide in game text explaining challenges and car parts.
Chapter 4

Architecture and Implementation

In this chapter the architecture of the game is detailed. First a general overview will be given, followed by a more detailed view of various important components. During our implementation of this architecture we were in some ways constrained to the decisions made by the prior group. Insofar as we desired to apply changes to it, the architecture that was already implemented could be extended on and modified quite well, so it was hardly constricting. Several implementation decisions are also given.

4.1 Architecture Overview

In this section we give an overview of the game architecture. We start out with the architecture we were given to work with initially. The deviations and additions to this we implemented will follow in a more detailed component based description in section 4.2.

4.1.1 Original Architecture

Main functions

Originally, the game consisted of two separate applications, namely the race game and the race designer. Both inherited from a base class. Functionally that was (reasonably expected to be) needed for both the race game and the designer was contained within a library. The library contained functionality such as a hierarchical spatial node system, 3D model classes used for displaying 3D models in game, networking and 2D graphics display and a few other smaller things. Every 3D game object was a node in a node-graph rooted in the world node. Using this, transformations could propagate through the tree, allowing for parented transformations of 3D objects. For physics, a 3D model could have a JigLibX collision model attached to it. JigLibX
being an external physics library. More on the role of JigLibX in the game is given below. 2D objects were implemented using a class supporting a lot of the XNA functionality for displaying sprites and text. HUD elements in the game were rendered from a specialized HUD class in the race game and simply rendered inside the main application in the case of the designer. The gameplay logic for the race game was handled by the RaceLogic class, which was updated from the main race game. This was mainly constrained to managing lap statistics. Another function fulfilled by the race game was reacting on user keyboard / gamepad input by moving the car. This was achieved by subscribing functions to an event listener class contained in the library. In the case of the race game, the functions subscribed implemented the movement of the car. The same event listener was used by the designer using mouse input events, rather than keyboard and gamepad events. Furthermore functionality for linking graphical models to JigLibX physics simulations was contained in the race game. Both applications communicated through a server-client setup. The race game representing the server and the designer representing the client. Changes made to the car in the designer were pushed onto a network and sent to the localhost, as the game runs on a single computer. When the race game received such a change, it updated the physical model of the car by pushing the changes through to JigLibX and updated the 3D graphics accordingly.

**JigLibX**

JigLibX is an open source library which works on the XNA-framework. It is ported from JigLib and is written in C# [1]. The Physics engine can be used for creating physics interactions between objects and models like collision and gravity in a game. JigLibX gives a solid base for creating a game. It shortens the developing time by letting programmers avoid the trial and error of that is required for building physics software from scratch. One can spend more time on making the game more robust or one can add an extra feature.

In the preliminary project the developers already spent time to find a suitable open source/ free software solution to make the prototype with. It would be a waste of time to redo survey and search for other possibilities. Because some of the project members already had some experience with JigLibX, we needed less time to accustom ourselves with unfamiliar source code.

JigLibX has an implementation of a car that gives a realistic behavior of a car. The car will skid when the car goes too fast around a corner. It uses real cars mechanic formulas to create car behavior, but some of the mechanics use a simplified formula. For example objects and cars have a linear acceleration instead of a logarithmic acceleration. And only the wheels contains some form of mass inertia. With realistic mass inertia the
car needs time and energy to reach top speed and needs brake and friction to slow the car down. If the car reaches top speed then it will maintain the top speed when player no longer presses the gas pedal, due to a lack of air and mechanical friction. The player would expect the car to slow down.

JigLibX source code is not made to be customized. It does not have sufficient comments and the main function has a few hundred lines of code and several functionalities. It is easy to get unpredictable and unrealistic car behavior when the source code or parameters are changed. The existing prototype uses brute force for collision detection. It slows the game down when the game always has to detect collision between all objects with moveable objects in the game world.

4.2 Component description

A more detailed implementation and architecture overview is given in this section. It includes a description of the main areas where we expanded or modified the preexisting architecture and/or implementation.

4.2.1 Client-Server model and Network

In this model, the racegame fulfills the server side role, while the designer is the client. The version of the game we started out with only supported one way, client-to-server messages. Specifically, only the changes that were applied to the car in the designer (client) were sent over to the racegame (server).

Before implementing any changes to the code, we had decided on several gameplay elements that required information to pass from the racegame to the designer. The minimap displayed on the designer screen is an example of this as the designer needs positional information from the racegame to translate to a 2d-position on a map. One way communication was inadequate for this and other gameplay features we had in mind, so we overhauled the networking to allow two way communication.

The one way communication approach made it possible to test the individual parts of the game separately in early stages of development. As time progressed, and the communication became more intricate, it often became necessary for both client and server to be running in order to test changes. In fact, the networking code has undergone several rewrites and refactorings during development, because test-runs proved it to not be general enough. For the interaction both sides of the game have to be able send and receive data. Because the racer and designer are using the same computer and same socket, some problems in the network occur. Some of the data packages are deformed when they arrive on the other side. Because the expected time to build a synchronized socket was more that a week and building it would be risky, we decided to make a workaround for the network. To lessen the
To detect the packages deformation, each packed is prepended with an integer header. This made it possible to send and detect different kind of packages. When the header is not recognized the package will be ignored. As a result, sending a single package once is not reliable. For reliable transfer a stream of packages is needed. For this reason synchronization of the selected car part and credits between racer and designer needs to be achieved. We implemented the sending of acknowledgements to attain this.

4.2.2 CS-model task delegation

Separating the tasks executed by both parts of the game was done in such a way that minimal network traffic was required. Sometimes this was overwritten to keep things neat. This was motivated in no small part by the difficulties we encountered with the networking. In practice this meant that on occasion functionality (e.g. scorekeeping) had to be transplanted from serverside to clientside. Some functionality is common to both server and client. Ultimately, the functionality was separated as follows:

Server:
- Running race game engine, including physics and 2d collision
- Running an adjusted JigLibX
- Process configuration changes received from designer
- Manage gamestate and gamestate changes
- Maintain credit count
- Displaying highscores
- Manage race progress and challenges

Client:
- Keeping score
- Handling name input, including reading and writing to file

Both:
- Playing audio
- Displaying 3d elements and HUD
- Switching language
- AFK reset
- LiveTimer

Each of these functions will be expanded on in their own sections below.
4.2.3 Racegame (server)

The race game engine, physics and 2D collision

For the racegame engine was implemented using Microsoft XNA. The external library JigLibX was used for physics, with some modifications to suit the needs of the game. More on this in below. A rudimentary positional check was already present in the game when we started. For gameplay purposes, it was necessary to check whether the car was in certain areas. This was used for checking whether the car passed checkpoints or the finish line. A simple Euclidian distance check between some 3D-point and the car was used, and given a distance threshold the cars presence within the region was determined. Since we wanted to implement bonus strips and other challenges requiring precise determination of the cars location this simple approach was to low on detail and could also be poorly tweaked and was somewhat difficult to visualize in game. For these reasons we implemented a new collision system, that only operates in 2D to keep it both simple and lightweight. The collision system operates on determining whether a point is contained within a quadrangle. In order to do so, a given quadrangle to check against is divided into two triangles on which a check is run. Given triangle ABC and point P, this check consists of calculating whether P is on Cs side of line AB, Bs side of AC and As side of CB. If all three conditions hold, P is contained in ABC.

Running an adjusted JigLibX

JigLibXs collision detection was a cause of performance drops. This is hardly surprising since collision detection is traditionally an area where one might expects a lot of operations are needed in real time. Our instance of JigLibX uses a simple brute force method to detect collision between all objects. It even calculated if there is collision between two immovable objects. By reduction the number of collision detections between immovable objects and removing a number of defensive if-statement checks that always turned out to be true we were able to alleviate some of the problems. The performance drop became acceptable, because frame rate stayed above 20 FPS. Even if the player tried to make the collision detection do a lot of work on purpose then it the game slows down less than a second. The high end system on which the game was to be played turned out to be easily capable of handling the workload.

If the car hits the track wall at high speed, it is possible for the car end up upside down or stuck. The wheel is turning in the air en JigLibX cannot rotate the car back to its upright position. To solve this problem we have implemented an unstuck function in the game to move the car back to the last known marker position in the track by pressing on start button on the wheel. To avoid exploitation of this function the speed of the car will be set
to zero when the position is reset.

**Process configuration changes from the designer**

JigLibX being deeply ingrained in the game, this function essentially changes the composition of the car by switching the current components with new ones, and announcing this change to JigLibX, by refreshing the necessary values. These values also needed resetting to default after each race.

**Manage gamestate and gamestate management**

We have implemented a gamestate to distinguish between the various phases of the game. These phases are: game start, intro, designer, race and high-score. The highscore state has several substates so inputting a team name, a detailed score view and comparing other players scores can be done consecutively in the same state. When certain in-game conditions (such as the car passing the finish line) are met, the gamestate has to be advanced, or possibly reset to prevent the game from being left running when the players have left. Both parts of the game use the gamestate, so it is important that they are both in the same gamestate. Hence, the gamestate is kept serverside, and the client may only change it by request. For example, when the game is in the start game state then touching the touch-screen will go to the introduction screen, but when the players are racing the game state will only be changed when the car finishes a lap or no input is detected for a while.

**Maintaining credit count**

For a long time this was handled clientside, since the spending and receiving of credits done by the player on that side. However, keeping this synchronized with the server proved troublesome in some rare cases. Therefore we elected to maintain the credit count serverside. Each credit spent by the client is now reported to the server, which then has to acknowledge the change before the in-game adjustments associated with the expense of a credit are made.

**Displaying highscores**

Since the racegame has the most screen real estate, we chose to have the comparison of the obtained score with other highscores on the server side. The storage of the obtained highscore is done clientside, so the highscore is now loaded from the hard disk where it was stored.
Managing race progress and challenges

This function can be summarized as the race logic class. It does bookkeeping and monitors race progress and the progression through challenges. In order to monitor this, indexed strips of 2D collision quadrangles (as described above) are used. Each challenge has its own unique set of quadrangle strips. The race logic class contains a reference to the HUD, so it can show the player what changes it detects.

A dedicated challenge manager is used for activating, deactivating and checking progress through the challenges. A challenge is activated once the car enters the activation quadrangle of a challenge, and deactivated if it enters the deactivation quadrangle. While a given challenge is active, the challenge manager updates the active challenges internal logic. The internal logic contains elements such as the amount of credits earned, telling the audiomanager to play certain cues, and maintaining internal timers. During this time, the challenge manager also updates the HUD elements that represent the challenges.

In order to keep track of the progression through the race, a strip consisting of indexed quadrangles is also laid out across the entire track. The index of the quadrangles ranges from 0 at the start of the race to about 140 at the end. When the car moves from a lower index to a higher one, the higher index is stored. When the car enters a quadrangle indexed lower than what is currently stored, the car is traveling in the wrong direction and the player is notified about this in the HUD.

A small number of the quadrangles covering the entire track also serve a secondary purpose: when they are entered, a message is sent to the client making it show the correct rallysign.

Late in development, we added a unstuck function, which uses the center point (just an approximation, not mathematically rigorous) of the quadrangles for resetting the cars position and facing direction.

4.2.4 Designer (Client)

Keeping score

Since we do not want the score to depend too greatly on the lap time and also do not want to reward driving very slowly in order to drive the score up, the scoring has largely been decoupled from time based considerations. Only a small bonus is awarded for good lap times.

The current score is calculated every tick of a dedicated clock, at the end of the race the total score is normalized to the total amount of ticks. The current score is calculated based on the configuration of the car. The various areas in which score can be obtained are also stored individually.
Name input

During the highscore state, a QWERTY keyboard is shown onscreen where the players can enter their team name. A profanity filter is in place, using a list of banned words downloaded from the Internet and extended with a range of Dutch words. The entered name and score are stored in a file on the hard disk. A separate file is created for the latest score. This is done so the server can construct the highscore list from two separate files to prevent read/write conflicts that would occasionally pop up during testing.

4.2.5 Both

Playing audio

RaceLibrary contains a static class AudioManager class. During startup time, this class loads audio cues that can be requested to be played from anywhere in the code. Since the vast majority of the games sound is based on the behavior of the car, the AudioManager contains a reference to the car, in order to produce engine and environment sounds.

Displaying 3D elements and HUD

This obvious need is handled differently by both parts. The racegame side is quite straightforward. First the 3D objects are rendered. 2D objects are deferred so they are drawn last. In the racegame all HUD elements are drawn inside a single class. When a given part of the game needs to draw a piece of the HUD, a reference to this class is used. Inside a designer draw call there are 3 different phases; first the 2D background elements of the screen and buttons are drawn, followed by all 3D objects, and finally the 2D overlays and text is drawn.

Switching language

Though language selection is handled clientside, text strings are displayed on both sides. All the onscreen strings are stored in an XML file, and both supported languages have a file. When the language is changed, these strings are refreshed from the appropriate file.

AFK reset

When players stop playing in middle of the game and go away, the game should automatically reset to the initial state after a given amount of time.\(^1\) Players that arrive afterwards thus do not start in the middle of a game. The game keeps a timestamp when the player has given his /her last input.

\(^1\)The acronym AFK is common Internet chatroom terminology and stands for ‘Away From Keyboard’
to the game. When the waiting time of one minute expires the game is going to reset.

LiveTimer

On several occasions during development we experienced a performance drop after new code was used. For example, after the first user test, the game faced some performance problems in the form of frame rate drops when the car was near the game challenges.

In order to identify the code responsible for this we built LiveTimer. With it could monitor the run times of various pieces of code during the race. LiveTimer was a class combining some of C#'s functionality for doing code timing. If the real-time execution time of a piece of code was to be tested, a static ‘start’ call could be made to the LiveTimer class using an identifier parameter. LiveTimer then pushed the identifier onto a stack, along with the time the call was made. When another ‘end’ call was made with the same identifier, the identifier was popped and the time spend on the stack measured. Using this construction we could measure nested functions execution time and display them on screen in a convenient way. LiveTimer helped the process of identifying and optimizing slow code immensely.
Chapter 5

Assets

Assets are all the visual and auditory components present in the game. None of the members of our development team were designers, and the assisting designers from the original teams role was not one of providing artwork. As a consequence, all the art we created is not of the same quality as some of the art that was already in there. Fortunately, the game was never intended to be a demonstration of graphics techniques. Equivalent reasoning applies to the audio portions we created. In this chapter the philosophy and development methods for 2D, 3D and audio assets will be given.

5.1 2D Assets

The original game provided a host of functions allowing for 2D graphics, which were only sparsely used. All 2D objects derived from the AHUDP- Drawable class, which provided transformation and translation classes. We added colorization and multi-language support to this class. The language support was relevant because the HUD text elements were also subclasses of AHUDPDrawable. Later, animated 2D-elements were also created as a subclass of AHUDPDrawable.

We wanted icons and on screen text to be clear and recognizable. In order to achieve this, the adage simpler is better or alternatively KISS\textsuperscript{1} applies. Because the implementation of the 2D graphics is straightforward in XNA and was well supported in the existing code, the only obstacle was to make sure the graphics were easy to understand. One way of achieving this was by accompanying every graphic with a short explanatory text. Secondly, we used color schemes of complementary colors; black, yellow, blue, red and greenish colors. This made game play icons instantly stand out on screen.

For creation of these 2D assets we used the usual suspects of software in this area; Adobe PhotoShop, the GIMP and Microsoft Paint for the most

\textsuperscript{1}Keep It Simple, Stupid!
5.2 3D Assets

Two members of the team had a small amount of experience creating three-dimensional models in respectively the Autodesk 3D Studio Max and Maya environments. After an instruction by one of the project assistants on some of the 3D content pipeline details that were unclear to us, we were thus able to place new 3D components into the game.

Amongst other things, user tests indicated the racetrack environment to be too empty (see section 3.3). We added several doodads to the race track to alleviate this. The ground texture of the racetrack and the surrounding areas was also given a bit of variation in both height and texture. Some of the 3D icons representing design choices were also found to be unclear, so we replaced them and added a caption as well.

5.3 Audio Assets

Originally there was no audio in the game. The reason we were given for the lack of audio is that there was insufficient time to fully implement audio. Indeed, the quality of a game suffers when only parts of a game contain audio, instead of it being present throughout, so audio should be given plenty of time.

When used correctly, sound enhances the gaming experience making the game more attractive and intuitive to play. In order to increase feedback from the game we decided to implement it. We started out introducing engine sounds, making it easier for the driver to identify the current engine choice. Tire choice was only visually noticeable by the width of the front tire axle with large tires having a wide axle and small tires a smaller one. In order to enhance the driving experience depending on the tire choice, we introduced skidding sounds. Small tires skid more and make the sliding of the car more noticeable.

To make the game more attractive, we added environment sounds. These are sounds that react according to the drivers input. The engine reacts according to the amount of gas the driver is giving, and the tires sound different when driving across different surfaces such as asphalt, grass and gravel. The driver can also hear when he or she hits a wall or a cone.

Sounds were also implemented into the designer user interface. The objective of these sounds is to provide feedback to the designer, making it clear that a change or upgrade has been executed and the vehicle has been updated. XNA is accompanied by the Cross-Platform Audio Creation Tool (XACT). This tool keeps track of each sound created and encloses the sound in a cue. The cue can then be accessed and even adapted through lines of
code. Adaptation of the sound can be heard for example as the engine reacting with the throttle or the variation of skid volume depending on the car speed.
Chapter 6

Final product

This chapter contains a description of the final product. First the race game and the race designer in their final form will be discussed. The chapter will conclude with a comparison with what we set out to create and the final result.

6.1 Race game

The race game now automatically launches full-screen on the appropriate screen and can be controlled by the Xbox360 steering wheel. It includes explanatory text on the different aspects of the game, which are shown prior to the actual racing. Obtained highscores can be compared with highscores of other players on the race game screen.

The race itself consists of a single lap across the Zandfoort track. Two bonus strips, a time trial and a slalom are placed on the track and they can be failed or completed by the player. Upon (partial) completion of a challenge, the players are awarded credits which can be used for upgrading the car. During the race, changes to the configuration of the car and upgrades can be made by the designer. Sounds produced by the car engine, bumping into objects, car configuration changes or challenge feedback are included.

6.2 Designer

Like the race game, the designer also automatically launches on the appropriate screen. It is controlled via touchscreen input. The same explanatory text is shown. During the racing phase, it shows a minimap of the cars’ position on the track and provides flashing rally signs which show information on upcoming parts of the track. The user is able to apply changes to the configuration of the car. The engine, wheels or bodywork material can be changed. A purely cosmetic choice of coloration for the car is also available. Upgrades can be applied to car parts by spending credits earned in
the race game. After the race is over, the player is able to check graphs of their obtained score in four different scoring areas. These areas are ease of production, environmental effect, maintainability of the car and mileage. A team name may be entered after the graphs have been shown. Team names are checked for profanity. Finally the credits are shown on in the designer.

6.3 Unachieved goals

Several of the goals outlined in section 2.2 were not achieved. The most prominent of which is the limited choices in parts available to the designer.

There are several reasons for this though the time constraint is the primary one. On a different level, the HUD of the designer needed large icons, if only because a user cannot be all that precise on a touch-screen. The nine choices plus their upgrades currently take up a large portion of the screen and implementing another interface for more parts would have been a large time drain. Additionally, having the changes reflected in JigLibX proved difficult enough with the diversity of parts we currently have. Adding more options would make this more complicated. Finally, we also began doubting the gameplay value of these additions during the play testing and development of the game.

The one-off design choices who's implementation we contemplated were strongly discouraged by the supporting staff, so we elected to drop them.

The ‘ideal line’-challenge was also dropped as it was merged into the bonus strip challenge, which proved difficult enough for players.
Chapter 7

Deployment

This chapter describes how the game is set up to be played in its final form. The various problems and their solutions are given below. Since the game is to be placed in the Science Centres game lab, we needed to make sure that game starts automatically and in such a way that players cannot access the underlying system provided by Tinker.

7.1 Set-up

The game runs on two screens; one large monitor for the race game accompanied by a touch screen for the designer part. Both screens are connected to a single computer that has two GPUs; one for each monitor. No keyboard or mouse is included in the final set up limiting a players access to the computer. The controls for race game is handled with a Microsoft Xbox360 wireless steering wheel.

7.2 Technical difficulties

7.2.1 Controller connectivity problems

During development, we used a wireless controller for testing, for the simple reason that the steering wheel and associated pedals were cumbersome to test with. We experienced some problems with the wireless controller shutting itself off (presumably to save battery power) after roughly 13 minutes of no input. We tried some elaborate ways of circumventing this to no avail. Fortunately we found out that the steering wheel does not disconnect automatically, at least not for the time span of roughly a working day that we tested it. Regardless, legacy support for a disconnecting controller remains. Should the game detect that the gamepad is no longer connected, the user is prompted to reconnect by pressing the connect button on the steering wheel.
7.2.2 Full screen problems

If the game runs in full screen mode, there is no way to close it down without a keyboard. Ideally, both games would run in full screen on their own screen, but this proved problematic. Windows offers very poor support for simultaneously running two games full screen on separate monitors. For one, processing power is allocated to the active screen which in our case is the designer game, since mouse (touch screen) input is made there. However, the race game is vastly more demanding of processing power. This lead to very poor performance for the race game. Luckily, C# offers a simple way of changing the process priority with a single line of code. With the priority set higher, the performance problem goes away. Secondly, when running two applications in full screen, and one is activated, the other one minimizes to the task bar. This lead to the race game disappearing to the taskbar as soon as the designer received any input, which was of course unacceptable. It seems to be outright impossible to run two applications in full screen at once. In addition to this, the XNA support for moving a game window by code proved to be somewhat unreliable, particularly when combined with full screen applications. For instance, moving non-active window activates it, minimizing the other full screen application.

7.2.3 Solutions to deployment problems

All this lead us to use a third-party scripting application called AutoIT, which has (amongst many others) the functionality to move windows around and start applications. The latter is important because the functionality offered by Windows to do this lead to a crash which seemed to be caused by the GPU not being able to handle the starting of two applications nearly simultaneously. The solution we finally arrived at, works by executing an AutoIT script at startup. This script first starts up the race game, in pseudowindowed mode and places it on the large monitor. That is, the game is in fact windowed, but the Windows title bar has been removed, giving a fake full-screen effect. After this the designer part is started by AutoIT, also in windowed mode. If it started in full screen it would either crash or be placed on the wrong monitor. The designer is then placed on the touch screen. A timer has been programmatically inserted to put the designer screen in full screen mode after all the steps described above have been completed. Since the user can not manipulate the race game other than through the steering wheel, and can only use the touch screen which has a full screen application on it, it becomes impossible for them to access anything else but the in-game features.
Chapter 8

Recommendations

The project took place over a period of 12 weeks and within this period we delivered a finished product. Since this product is a piece of software there is always room for improvement. In this chapter we will discuss which parts of the game can be improved and how they can be improved.

8.1 Physics

The Jiglibx physics engine provided a quick and easy way to get the physics up and running but it proved to be a major setback when trying to adapt it in order to make it more realistic. One option would be to develop a physics engine from scratch that fit the game completely. That would result in a more realistic driving experience and also more (accurate) data that would be used to define the players score. Some examples of added data could be:

- Tire wear
- Fuel consumption
- Measuring CO₂ emission

8.2 3D-Models

One of the features we worked on was adding models to the track, making it less empty and more appealing to the audience. This eye-candy feature could be taken to the next level and a replica of the Zandvoort racing track could be modeled and displayed in the game.

8.3 Multiplayer

The game has been developed and deployed in a two-player setting. If a second setting is ever deployed, they could be both linked so two pairs could
compete against each other. Multiplayer support will require a major rewrite in the network structure since the current setting is already a client-server structure.

8.4 Multiple track and car choice

Different tracks is a simple manner to introduce varying difficulty to the game. A track with lots of turns and corners provide a higher difficulty level for the driver while a track with variation provide a bigger challenge for the designer. Tracks from other countries give the opportunity for foreign visitors to race at their home track and Dutch visitors to race abroad. Adding other vehicles to the game will enhance the Serious Gaming experience since design choices will have different impact on different vehicles. New vehicles could be the Eco Runner, DUT 10 or the Superbus.
Bibliography

Appendix A

A.1 Assignment
A.2 Plan of Approach
A.3 Game Design Document
A.4 Simplified object diagram of the game